In the Specification:

Please replace paragraph [0006] of the application with the following amended paragraph:

[0006] Trust to remote humans or devices, interacting over electronic networks, has two components. The first component is identification and verification of the parties at the beginning of the communication session (mutual authentication). The second component is associated with trust to information transferred during the communication session over untrusted communication media (communication lines). It includes the following specific requirements - confidentiality (none can read the message except the intended recipient), integrity (none altered, tampered with, or modified the message against the original), and non-repudiation (the sender ~~can not~~ cannot deny the fact of having sent the message).

Please replace paragraph [0038] of the application with the following amended paragraph:

[0038] For instance, in FIG. 1, SRK1 appears in ASK 1013 at the time mark "0 minute", and at the moment that time mark 1 minute LT1 of SRK1 has expired, though SRK1 remains inside ASK 1013. SRK Generator 1005 at this moment generates [[SRK 2]] SRK2 and places it into ASK 1013. By the time mark 2 minutes [[SRK 2]] SRK2 LT1 has expired, even though [[SRK 2]] SRK2 remains inside ASK 1013. Again, at this time SRK Generator 1005 generates and places into ASK 1013 [[SRK 3]] SRK3, which LT1 becomes expired at the 3 minutes mark. This procedure is periodically repeated as long as SRK Generator 1005 is on. Client 1 1003 and Client N 1009 made a connection request during the time interval between time mark 4 minutes and time mark 5 minutes, since SRK Generator 1005 began generating SRK 1011 and filling them into ASK 1013. The only SRK 1011 not yet expired LT1 in ASK 1013 during this time interval is [[SRK 5]] SRK5. Therefore, [[SRK 5]] SRK5 is used to establish communication sessions with these clients. Similarly, Client 2 requested a communication session between time mark 8 minutes and time mark 9 minutes, whereas Client N-1 requested a communication session between time mark 1 minute and 2 minutes. Hence, the SRK 1011 used to establish these communication sessions are, respectively, [[SRK 9]] SRK9 and [[SRK 2]] SRK2.

Please replace paragraph [0039] of the application with the following amended paragraph:

[0039] Once LT1 is expired, the server generates and places into ASK 1013 another SRK 1012, which LT1 is just started. SRK 1011 second life time LT2 defines the life time inside the limited size ASK 1013. The maximum size of ASK 1013 can be characterized with the parameter Nmax which indicates maximum number of SRK 1011 in ASK 1013 possible (for instance, Nmax = 5 in FIG. 1). Typically, LT1 < LT2, and in the most preferred embodiment LT1 can be derived as LT1 = LT2 / Nmax. Without sacrificing any generality limitations of TILSA, LT2 was chosen, for example, to be equal to 5 minutes in FIG.1. Then, LT1 according to the formulae presented above, is equal to 1 minute. After LT1 expired, for any given SRK 1011, the key has LT2-LT1 time remaining to support communication session threads having been initiated during LT1. Once LT2 expired, SRK 1011 is removed from ASK 1013, effectively canceling any further participation of this particular SRK 1011 in the parties' communication session engagements. Certainly, each SRK 1011 can be used to originate multiple threads of communication sessions with each Session Elapsed Time (SET) less or equal to LT2 - LT1. However, SET = LT2 - LT1 is the preferred embodiment. Without sacrificing any generality limitations of TILSA, SET = 4 minutes in FIG. 1. Taking SRK5 in FIG. 1 as an example of any SRK 1011 genesis, one can note that [[SRK 5]] SRK5 is the last key to fulfill ASK 1013 to its maximum size Nmax = 5, and [[SRK 5]] SRK5 appears inside ASK 1013 at the 4-minute mark, since SRK Generator 1005 began generating SRK 1011 and filling them into ASK 1013. Then, during [[SRK 5]] SRK5 LT1 = 1 minute, the key can be engaged into initiating multiple communication session threads with the clients requesting connections. From time mark 5 minutes, and until time mark 9 minutes, SRK5, in accordance with SET = 4 minutes, is kept inside ASK 1013 available to support communication session threads started during [[SRK 5]] SRK5 LT1. During this particular time interval, from time mark 5 minutes to time mark 9 minutes, [[SRK 1]] SRK1, [[SRK 2]] SRK2, [[SRK 3]] SRK3, and [[SRK 4]] SRK4 in ASK 1013 are being gradually replaced every minute by SRK6, [[SRK 7]] SRK7, [[SRK 8]] SRK8, and [[SRK 9]] SRK9, respectively. Eventually, at time mark 9 minutes SRK5 is canceled, ultimately being replaced by SRK10.

Please replace paragraph [0043] of the application with the following amended paragraph:

[0043] Since SRK 1011 is sent to the client as the first message, the logic located on the server and on the client sides generates a series of messages having been sent from the server to the client, and back to the server with the following Key Encryption/Decryption Iterative Algorithm (KEDIA). FIG. 3 is a graphic illustration of the Key Encryption/Decryption Iterative Algorithm (KEDIA) according to the present invention. In step 1 3005, client 3002 sends a connection request to server 3001 over communication network 3003. In step 2 3006, SRKi (with the currently active [[LT1]] LT1i – between time mark i-1 minutes and time mark i minutes) is sent to client 3002, and stored there, initiating the communication interface. In step 3 3007, client 3002 enters the user name, the user password, and the server password, if it is a user at the client platform 3002, or the host name, the host ID, and the server password, if it is the client platform (the peer). In step 4 3008, the user name (or the host name) is hashed, encrypted with SRKi and sent to server 3001, while the user password (or the host ID) and the server password were not sent, remaining at client 3002.

Please replace paragraph [0044] of the application with the following amended paragraph:

[0044] In step 5 3009, server 3001 checks the validity of the user name (or the host name), obtained in the step 4, through the database to which it is connected. The session is terminated, if the user name (or the host name) is not valid. Otherwise, server 3001 in step 3009 sends [[DRK 1]] DRK1 encrypted with SRKi to client 3002, where [[DRK 1]] DRK1 is decrypted with SRKi, and stored at client 3002. During the same step 3009, client 3002 sends a [[DRK 1]] DRK1, which is converted to its hash equivalent and encrypted with [[DRK 1]] DRK1, to server 3001. This message confirms to server 3001 that client 3002 obtained and decrypted [[DRK 1]] DRK1, and it is ready for receiving another secret key. In step 6 3010, server 3010 first decrypts hashed [[DRK 1]] DRK1, received in step 5 from client 3002, with [[DRK 1]] DRK1. If [[DRK 1]] DRK1 is correct, server 3001 sends [[DRK 2]] DRK2 encrypted with [[DRK 1]] DRK1 to client 3002, where [[DRK 2]] DRK2 is decrypted with [[DRK 1]] DRK1, and stored at client 3002. Otherwise, the communication session is terminated. During the same step 6 3010, client 3002 sends a [[DRK 2]] DRK2, converted to its hash equivalent, and

encrypted with [[DRK 2]] DRK2, to server 3001. This message confirms to server 3001 that client 3002 obtained and decrypted [[DRK 2]] DRK2, and it is ready for receiving another secret key.

Please replace paragraph [0050] of the application with the following amended paragraph:

[0050] Therefore, an offline attack is senseless, as the intruder going backward through steps 3013, 3010, 3009, and 3008 could find DRKn-2 DRKn-3, …, [[DRK 1]] DRK1, and eventually SRKi, which are all only one-time session random keys, and they ~~can not~~ cannot be reused. Certainly, the intruder could further decrypt the user name; however, this is not regarded as a secret. The time DRKn-1, operating during the client/server communication session, is excruciatingly small for attempting an online computer processing attack. Even assuming this attack successful, all, the intruder could do is to send to client 3002 an incorrect authentication signal, which will be visualized in the user's session GUI, but would never take effect in the actual system. This is because the authentication signal "go/no" enables functionality through the server 3001 side logic.

Please replace paragraph [0052] of the application with the following amended paragraph:

[0052] FIG. 4 is a graphic illustration of the KEDIA algorithm. This is a typical message encryption at the server and its decryption at the client, applying along with encryption and decryption procedures one of Byte-Veil-Unveil (ByteVU), Bit-Veil-Unveil (BitVU), or Byte-Bit-Veil-Unveil (BBVU) algorithms, according to the present invention. Step 6 3010 has been chosen as a typical message example in the KEDIA algorithm. According to FIG. 3, during this step, server 3001 sends [[DRK 2]] DRK2 encrypted with [[DRK 1]] DRK1 to client 3002, where [[DRK 2]] DRK2 is decrypted with [[DRK 1]] DRK1, received by client 3002 in the previous step 3009 from server 3001. In FIG. 4, step 3010 is split for clarity into two parts 4001 and 4002, which are related to preparing the message at server 3001, and treating the received message at client 3002, respectively. Blocks 4003, 4005, 4007, and 4009 depict the process the message is going through, before it is sent to client 3002. [[DRK 2]] DRK2 (for instance, 16 bytes long, secret key to be used with a block-cipher encryption algorithm) is supplied by Server

DRK Generator 2005 (see FIG. 2) 4003. In the following step 4005, server 3001, already having identified who claims to be the user on the client platform, (or what is the claimed client platform host name), extracts the respective user password (or the client host ID) from the database 3004 attached to server 3001. Eventually, according to block 4007, server 3001 uses this information to trigger operation of one of ByteVU, BitVU, or BBVU algorithms, having been chosen by a particular security system, considering security requirements vs. cost trade-offs (time of operations, CPU power of client/server platforms, and the network throughput). As a final result 4009, the conversion array, containing disassembled [[DRKj]] DRK2 bytes, or bits, or the combination thereof, gets encrypted with DRK1, and sent to client 3002.

Please replace paragraph [0053] of the application with the following amended paragraph:

[0053] Part 4002 of step 3010, related to the received message treatment at client 3002, is expanded by the series of blocks 4004, 4006, 4008, and 4010 in FIG. 4. According to block 4004, client 3002 decrypts the conversion array with [[DRKj-1]] DRK1, stored by client 3002 from the previous message 3011 from server 3001. Then, client 3002 supplies the user password (or the client host ID) which was entered into the KEDIA algorithm at step 3 3007 (see FIG. 3), enabling reassembling of [[DRK 2]] DRK2 from the decrypted conversion array 4006. As it is shown in block 4008, the operation is triggered for one of ByteVU, BitVU, or the BBVU algorithms, having been chosen on the client side the same one, as on the server side. Eventually, according to block 4008, either the message bytes, or bits, or the combination thereof, get reassembled, and finally, as it is shown in block 4010, [[DRK 2]] DRK2 is reconstructed to its original form.

Please replace paragraph [0061] of the application with the following amended paragraph:

[0061] FIG 7 is a block diagram of the Byte-Bit-Veil-Unveil (BBVU) algorithm according to the present invention. Block 7001 shows DRKj, where each byte is separated from a neighboring byte with a vertical bar. Without sacrificing any generality limitations of the BBVU algorithm, DRKj is assumed to be a 16-bytes key in FIG. 7. The user password (or the client host ID), supplied by server 3001 in a hashed form, plays a seed role for Server Sequential

Random Number Generator (SRNG) 7002. SRNG 7002 generates a random sequence of 16 integers, and then the server's Sequential Direct Bit Position Scrambler (SDBPS) 7006 scrambles all bit positions in the veiled byte 7010. SDBPS 7006 generates a random series of non-repeating eight digits within the range from 1 to 8, for each of SRNG 7002 integers in the sequence. In other words, the password (or the client host ID), the SRNG 7002 sequence of integers, and the series of digits generated by SDBPS 7006 are uniquely associated. Applying the same seed (the user password, or the server host ID, in a hashed form) will result in the same sequence of integers generated by SRNG 7002, and the same series of digits generated by SDBPS [[7006]] 7007 for each integer in the sequence.

Please replace paragraph [0063] of the application with the following amended paragraph:

[0063] At client 3002, the encrypted conversion array is decrypted with DRKj-1, saved at client 3002, from the previous server message (step 3011 in KEDIA, FIG. 3). Then the procedure, a reversed one as compared to that which is described above for the BBVU algorithm on server 3001 side, is applied. The user password (or the client host ID), saved at the client platform in step 3007 of the KEDIA algorithm (see FIG. 3), is supplied in a hashed form as a seed to Client Sequential Random Number Generator (SRNG) 7005 identical to the one on the server 3001 side. This password (or host ID) triggers SRNG 7005 to generate the same sequence of integers as on server 3001 side before 4, … . Client Sequential Reverse Bit Position Scrambler (SRBPS) 7003 generates the reversed series of digits for each integer, as compared to its server counterpart SDBPS 7006. For instance, for the first integer 4, SRBPS 7003 generates the reversed series 2, 6, 1, 5, 4, 8, 7, and 3, which allows the logic placed on client side 3002 to restore bits in the original order for the 1st byte of DRKj~~[[—2]]~~ means that the 2nd bit of the scrambled byte will become the least significant bit in the restored 1st DRKj byte, and so on, until 3, the last digit in the series, is reached, indicating that the 3rd bit in the scrambled byte will become the most significant bit in the restored 1st byte. Meanwhile, integer 4 points to the 4th position in section 7008 of conversion array 7007, where the 1st DRKj byte has been veiled. The same procedure continues, until all byes of DRKj 7001 and their respective bits are returned to their original positions. This completes the reassembling procedure of the BBVU algorithm to restore DRKj at client 3002.

Please replace paragraph [0064] of the application with the following amended paragraph:

[0064] At this time it is important to note that the ByteVU, BitVU, and BBVU algorithms, disclosed above, require assessment of security of these algorithms against possible computer processing attacks now and in the future. Table 1 below presents a summary of this assessment. SRNG 5002, 5003 (FIG. 5), [[600]] 6002, 6003 (FIG. 6), and 7002, 7003 (FIG. 7) generate integers pseudo-randomly, as well as SDBPS 7006 and SRBPS 7003 (FIG. 7). Hence, probabilities of veiling each byte and bit inside a Conversion Array (CA) for each algorithm can be viewed as independent ones. Best microprocessors achieved ~1 GHz clock rate barrier by the beginning of the 21st century. Previously, forecasting allowed for at least 25 – 35 years, until the clock rate would reach ~ (100 – 1000) GHz. Thus, currently available ~1E10 instructions per second could reach ~ (1E12 - 1E13) instructions per second in a distant future, (assuming microprocessor RISC pipelined architecture with up to 10 stages per cycle). A very conservative assumption is made that the attacking computers have 100% efficiency of their CPU utilization during an attack. In other words, testing each possible combination of all bytes, bits, or the combination thereof , of a veiled message in CA will consume only one microprocessor instruction.

Please replace paragraph [0065] of the application with the following amended paragraph:

[0065] Column 1001 in Table 1 presents particular geometries of CA chosen in each algorithm for the assessment. Column 1002 gives the bit size of each algorithm CA for every geometry selected in 1001. Column 1003 presents the total number of pseudo-random integers generated by SRNG of each algorithm with respect to the geometries chosen in 1001. Column 1004 introduces probability models for each algorithm with respect to the geometries of CA chosen in 1001. Every position in 1004 gives probability to estimate the entire combination of veiled bytes, bits, or the combination thereof, for each algorithm, under given geometry of CA in 1001. Column 1005 presents for each CA its transit time, given the slowest standard modem of 28.8 kbps (kilobits per second) of contemporary networks (for example, the Internet). Column 1006 presents assessed time, required for a brute force attack now and in a distant future, for each algorithm and their respective geometries of CA chosen in 1001. Column 1007 presents an

approximate time for one advanced microprocessor ([[1GHZ]] 1GHz /100GHz) instruction now, and in a distant future.

Please replace paragraph [0069] of the application with the following amended paragraph:

[0069] Messages sent to the client and received at the server are numbered in 8000. Key functional message destinations on the server side are in 8001, and on the client side they are in 8016. For each message received at the server, its content description is in 8003, whereas for each message received at the client, its content description is in 8014. Similarly, for each message sent from the server, its content description is in 8002, whereas for each message sent from the client, its content description is in 8015. The choice of any one of ByteVU, BitVU, or BBVU algorithms to be used in the MEDIA protocol and the parameters of the respective conversion array are in 8006 for the server side, and in 8010 for the client side. Seeds, having been used to trigger SRNG (Sequential Random Number Generator), are in 8007 for the server side, and they are in 8009 for the client side. Which direction a particular MEDIA message is sent towards, is in 8008. The ByteVU algorithm conversion array parameters, chosen in FIG. 8A and FIG. 8B (10 sections with 25 bytes size of each), give extremely high security protection against online and offline intruding attacks, even for one MEDIA message as it was shown above. Therefore, it is practically justifiable to reduce iterations in the KEDIA algorithm by limiting DRKn in FIG. 3 to [[DRK 2]] DRK2 only. It saves client/server platforms CPU and network resources, while keeping a very high security level.

Please replace paragraph [0071] of the application with the following amended paragraph:

[0071] The server (logic on the server side in this exemplary case could be implemented in the Java servlet technology) replies in message 4 with [[DRK 1]] DRK1 2012 (FIG. 2) bytes veiled with the ByteVU algorithm, triggered by the server, supplying the hashed password of the assumed user as a seed. The resulting ByteVU conversion array is encrypted with the SRK and sent to the client. The client, having known the SRK and the user password, entered into the GUI in the previous message 3, decrypts the conversion array and reassembles [[DRK 1]] DRK1 bytes. In message 5, from the client to the server, hashed [[DRK 1]] DRK1 bytes are veiled with

ByteVU algorithm, triggered by the user password, stored at the client earlier in step 3 (FIG. 8B), and converted to its hash equivalent. Then, the conversion array is encrypted with [[DRK 1]] DRK1 and sent to the server, which decrypts the conversion array with [[DRK 1]] DRK1, and triggers ByteVU with the hashed password of the assumed user, taken from the database attached to the server. If the hashed [[DRK 1]] DRK1 is correct, reassembled in this way, it is actually the authentication signal from the client to the server, as nobody except the client knows the user password used to trigger the ByteVU algorithm when receiving message 4, and sending message 5.

Please replace paragraph [0072] of the application with the following amended paragraph:

[0072] If [[DRK 1]] DRK1 is incorrect, the MEDIA protocol is terminated by the server sending a "no" authentication message (or an error message: "user password is incorrect") to the client, encrypted with SRK. Otherwise, the server sends to the client message 6 containing [[DRK 2]] DRK2, which bytes are disassembled by the ByteVU algorithm, triggered by the user hashed password, used as a seed for SRNG 5002 (FIG. 5). Then, the conversion array is encrypted with [[DRK 1]] DRK1 and sent to the client, where it is decrypted with [[DRK 1]] DRK1 stored at the client from the previous message 5, and [[DRK 2]] DRK2 bytes get reassembled by the ByteVU algorithm, triggered by the user password, stored at the client earlier in step 3 (FIG. 8B). The client replies to the server with message 7, sending to the server hashed [[DRK 2]] DRK2, which bytes are veiled by the ByteVU algorithm, triggered by the user password, stored at the client in the previous message 3, and converted to its hash equivalent. The server decrypts message 7 from the client with [[DRK 2]] DRK2, and reassembles the hashed [[DRK 2]] DRK2 bytes with the ByteVU algorithm, triggered by the user password, taken from the attached to the server database, and converted to its hash equivalent. If [[DRK 2]] DRK2 is correct, the server sends to the client message 8 with [[DRK 2]] DRK2, which bytes are disassembled with the ByteVU algorithm, triggered by the server password. Otherwise, if [[DRK 2]] DRK2 is not correct, the MEDIA protocol is terminated. The conversion array of the ByteVU algorithm in message 8 is encrypted with [[DRK 2]] DRK2 and sent to the client.

Please replace paragraph [0073] of the application with the following amended paragraph:

[0073] The client, receiving message 8 from the server, decrypts it with [[DRK 2]] DRK2, and reassembles the hashed [[DRK 2]] DRK2 bytes with the ByteVU algorithm, triggered by the server password, stored on the client side in message 3. Then, the client compares the decrypted and reassembled [[DRK 2]] DRK2 with [[DRK 2]] DRK2 from the previous message 6. If they are the same, it is viewed by the client as the authentication signal from the server, because only the client and server share the server password. Hence, it was the only server, which could send the last message 8 to the client. Now, as the trust is established by the client to the server, the client sends to the server message 9 with hashed [[DRK 2]] DRK2, which bytes are disassembled with the ByteVU algorithm, triggered by the server password, stored on the client side in message 3, and converted to its hash equivalent. Eventually, the conversion array of the ByteVU algorithm is encrypted with [[DRK 2]] DRK2 and sent to the server. The server, having received message 9 from the client, decrypts it with [[DRK 2]] DRK2, and reassembles the hashed [[DRK 2]] DRK2 bytes with the ByteVU algorithm, triggered by the hashed server password. If [[DRK 2]] DRK2 is correct, it is viewed by the server as a second authentication factor from the client (the client confirmed the server password), in addition to the first factor, having been checked in the message 6 from the client (the client confirmed the user password).

Please replace paragraph [0074] of the application with the following amended paragraph:

[0074] This completes the mutual authentication of the client/server pair according to the MEDIA protocol, and the server is now ready to make an authentication decision. In the end, the server sends to the client message 10, which has either a "go" authentication signal, assuming [[DRK 2]] DRK2 in message 9 from the client was correct, or an error message: "the server password is incorrect", assuming [[DRK 2]] DRK2 in message 9 from the client was incorrect. Each signal byte is disassembled with the ByteVU algorithm, triggered by the user password from the database, attached to the server, and then the conversion array of the ByteVU algorithm is encrypted with [[DRK 1]] DRK1 and sent to the client in message 10. Having received the message 10, the client decrypts it with [[DRK 1]] DRK1, stored at the client platform during

message 4, and reassembles the signal bytes with the ByteVU algorithm, triggered by the user password, stored at the client side in message 3.

Please replace paragraph [0075] of the application with the following amended paragraph:

[0075] This effectively completes the entire MEDIA protocol of the client/server communication session as presented in FIG. 8A and FIG. 8B. As one can see, authentication credentials (the user password and the server password in this particular embodiment) have never passed through communication lines in any form. Also, the client/server mutual authentication has been completed within the MEDIA protocol, as well as the exchange of FSK (Final Secret Key, which is [[DRK 2]] DRK2 in this particular embodiment) having been performed within the client/server pair. The server password and the user password enable secure mutual authentication, according to the MEDIA protocol architecture. At the same time, they are both playing a role of a strong two-factor authentication of the client at the server platform.

Please replace paragraph [0077] of the application with the following amended paragraph:

[0077] GUI 9003 has several operation modes 9009: login session mode 9010, account set-up mode 9011, user password reset mode 9012, and server password reset mode 9013. Login session 9010 is the default operation mode. The user enters the user name in window 9004, the user password in window 9005, and the server password in window 9006. The user has a choice to enter alphanumeric characters, or their echo dots for security reasons by toggling button 9014. The session elapsed time clock 9007 visualizes this value to the user, and signals communication session termination once the session time has expired. After the authentication credentials are all entered into 9004, 9005, and 9006, the client indicates login button 9008, which completes step 3 3007 in FIG. 3, or message 3 in FIG. 8B. Then the other steps of the MEDIA protocol are initiated. Stoplight 9001 turns yellow, when button 9008 is indicated, signaling the MEDIA protocol is in progress for the first authentication factor (the user password) examination. Message 8 in FIG. 8B, having arrived at the client, initiates stoplight 9001 to change color from red at the beginning of the session to green, once it is checked by the client placed logic that

[[DRK 2]] DRK2 delivered in the message 8 is identical to [[DRK 2]] DRK2, delivered in message 6.

///